



**Instituto Politécnico Nacional**  
**Escuela Superior De Computo**  
**Sistemas en Chip**



### **Práctica 3**

Nombre de los integrantes:

García Quiroz Gustavo Iván  
Bejarano García Owen Uriel

Grupo: 7CV3

Nombre del Profesor: Miguel Ángel Castillo Martínez

Fecha De Entrega: 03/06/2025

# **1 Índice**

2	Introducción .....	1
3	Materiales y Equipo Utilizado.....	2
3.1	Hardware .....	2
3.2	Software.....	2
4	Desarrollo de la Práctica.....	3
4.1	Diagrama del Circuito.....	3
4.2	Código Utilizado.....	3
4.3	Procedimiento .....	6
5	Resultados.....	6
5.1	Observaciones Clave .....	13
6	Conclusiones .....	14
7	Referencias.....	16

## 2 Introducción

El control de velocidad en motores de corriente directa (DC) es una aplicación fundamental en sistemas embebidos y automatización. Una de las técnicas más eficientes para lograr este control es la Modulación por Ancho de Pulso (PWM), la cual permite variar la potencia entregada al motor sin perder eficiencia energética. En esta práctica, se implementó un sistema de dosificación de potencia mediante PWM utilizando un microcontrolador Arduino, donde se establecieron 10 niveles de potencia (del 0 al 9) ajustados experimentalmente para garantizar un rango de operación óptimo del motor.

El principio del PWM se basa en variar el ciclo de trabajo (*duty cycle*) de una señal cuadrada, donde un ciclo de trabajo del 0% mantiene el motor apagado, y del 100% lo hace girar a su máxima velocidad. Sin embargo, en la práctica, los motores tienen un umbral mínimo de voltaje necesario para iniciar el movimiento (debido a fuerzas de fricción e inercia), así como un límite máximo donde incrementar el PWM ya no produce cambios significativos en la velocidad. Estos valores fueron determinados experimentalmente en el laboratorio, definiendo así los niveles de potencia utilizados en el sistema.

La comunicación UART (Serial) se empleó como interfaz de usuario, permitiendo seleccionar los niveles de potencia mediante comandos numéricos. Esta técnica no solo es útil para motores, sino que también es aplicable en regulación de luminosidad en LEDs, control de servomotores y sistemas de conversión de potencia. La práctica demostró cómo el PWM, combinado con un microcontrolador, ofrece una solución versátil y eficiente para el control preciso de dispositivos electromecánicos.

### **3 Materiales y Equipo Utilizado**

#### **3.1 Hardware**

- Arduino Uno (ATmega328P) o equivalente.
- Motor DC 6-12V.
- Puente H L298N.
- Fuente externa de 5V-12V para el motor.
- Protoboard y Cables Dupont
- Osciloscopio

#### **3.2 Software**

- Arduino IDE.
- Monitor Serial integrado en Arduino IDE.

## 4 Desarrollo de la Práctica

El desarrollo de la práctica se centró en implementar un sistema de control de velocidad para un motor DC mediante PWM (Modulación por Ancho de Pulso), utilizando un Arduino como unidad de procesamiento y comunicación. A continuación, se detallan los aspectos clave del proceso experimental.

### 4.1 Diagrama del Circuito

El circuito constó de un Arduino UNO (basado en el ATmega328P), un motor DC de baja potencia, un driver L293D para amplificar la señal de control y una fuente de alimentación externa para evitar sobrecargar el puerto USB. Las conexiones se realizaron de la siguiente manera:

- **Salida PWM del Arduino (Pin 6):** Conectada al pin de entrada del driver L293D (IN1).
- **Pines de control del driver (ENABLE, IN1, IN2):** Configurados para permitir el giro en un solo sentido.
- **Motor DC:** Conectado a los pines de salida del driver, con un diodo de protección para evitar picos de voltaje inverso.
- **Alimentación:** Se usó una fuente externa de 6-12V para el motor, mientras que el Arduino se alimentó vía USB.

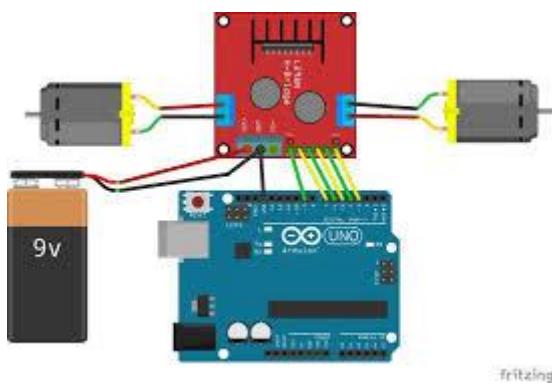


Figura 1 Diagrama esquemático del circuito (aproximación, la diferencia es que nosotros sólo usamos un motor)

### 4.2 Código Utilizado

El programa en Arduino se diseñó para recibir comandos por comunicación serial (UART) y ajustar el ciclo de trabajo del PWM según 10 niveles predefinidos (0-9). La lógica principal incluyó:

- Configuración inicial del Timer0: Se utilizó el modo Fast PWM de 8 bits ( $\text{WGM02:0} = 011$ ) para generar una señal PWM en el pin PD6 (OC0A). Se configuró el registro TCCR0A en modo no inversor ( $\text{COM0A1} = 1$ ,  $\text{COM0A0} = 0$ ), lo que hace que la señal PWM se limpie en coincidencia de comparación y se establezca en BOTTOM. El prescaler se estableció en 64 para controlar la frecuencia de la señal.
- Mapeo de niveles PWM: Los niveles del 1 al 9 se asignaron a valores de OCR0A calibrados experimentalmente. El nivel 1 se mapeó a un duty cycle del 30% (aproximadamente  $\text{OCR0A} = 77$ ), que representa el mínimo necesario para provocar movimiento visible del motor, y el nivel 9 al 100% ( $\text{OCR0A} = 255$ ), evitando saturación.
- Comunicación serial por UART: A través de interrupciones de recepción (`USART_RX_vect`), se interpretaron caracteres ASCII del '0' al '9' recibidos por UART. El nivel 0 apaga el motor (duty cycle 0%), y los niveles del 1 al 9 ajustan el PWM dinámicamente según el valor recibido. El motor se controla completamente desde la rutina de interrupción.

```
1 #include <avr/io.h>
2 #include <avr/interrupt.h>
3 #include <stdint.h>
4
5 #define FOSC 16000000UL
6 #define BAUD 115200
7 #define MYUBRR ((FOSC / 16 / BAUD) - 1)
8
9 // Estructuras para registros de configuración del Timer0
10 typedef struct {
11     uint8_t WGMn : 2;
12     uint8_t      : 2;
13     uint8_t COMnB : 2;
14     uint8_t COMnA : 2;
15 } TCCRnA_t;
16
17 typedef struct {
18     uint8_t CSn : 3;
19     uint8_t WGMn : 1;
20     uint8_t      : 2;
21     uint8_t FOCnB : 1;
22     uint8_t FOCnA : 1;
23 } TCCRnB_t;
```

```

25 volatile TCCRnA_t *TCCR0A_ = (TCCRnA_t *)0x44;
26 volatile TCCRnB_t *TCCR0B_ = (TCCRnB_t *)0x45;
27
28 // UART Init
29 void USART_Init(unsigned int ubrr) {
30     UBRR0H = (unsigned char)(ubrr >> 8);
31     UBRR0L = (unsigned char)ubrr;
32     UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0); // RX, TX y habilita interrupción de recepción
33     UCSR0C = (1 << USBS0) | (3 << UCSZ00); // 8 bits de datos, 1 bit de stop
34 }
35
36 void USART_Transmit(unsigned char data) {
37     while (!(UCSR0A & (1 << UDRE0)));
38     UDR0 = data;
39 }
40
41 // Configurar Timer0 para PWM Fast Mode
42 void setup_pwm_timer0() {
43     DDRD |= (1 << PD6); // PD6 / OC0A como salida
44
45     TCCR0A_>WGMn = 3;      // Fast PWM (WGM00 y WGM01 = 1)
46     TCCR0A_>COMnA = 2;    // Clear OC0A on Compare Match, set at BOTTOM (modo no inversor)00.
47     TCCR0B_>WGMn = 0;      // WGM02 = 0 (ya está en modo Fast PWM de 8 bits)
48     TCCR0B_>CSn = 4;      // Prescaler = 64 (CS02:0 = 011)
49
50     OCR0A = 0; // Inicialmente 0% duty cycle
51 }

```

```

53 // Interrupción al recibir datos por UART
54 ISR(USART_RX_vect) {
55     unsigned char received = UDR0;
56
57     if (received >= '0' && received <= '9') {
58         uint8_t valor = received - '0'; // Convertir ASCII a número
59
60         if (valor == 0) {
61             OCR0A = 0; // Apagar motor
62         } else {
63             // Duty mínimo del 30% y máximo 100%
64             OCR0A = ((valor - 1) * (255 - 77) / 8) + 77; // 77 ≈ 30% de 255
65         }
66     }
67 }
68
69 int main(void) {
70     cli(); // Desactivar interrupciones globales
71     USART_Init(MYUBRR);
72     setup_pwm_timer0();
73     sei(); // Activar interrupciones globales
74
75     while (1) {
76         // El motor se controla desde la ISR
77     }
78 }

```

Figura 2. Código utilizado

### 4.3 Procedimiento

El experimento se dividió en tres etapas:

1. **Pruebas iniciales:** Se determinó el valor PWM mínimo (ej: 28) necesario para vencer la inercia del motor, verificando que este respondiera sin carga mecánica.
2. **Rango operativo:** Se incrementó el PWM en pasos del 10% hasta identificar el valor máximo (ej: 250) donde la velocidad ya no aumentaba significativamente.
3. **Validación:** Se asignaron los 10 niveles equidistantes entre los valores mínimo y máximo, y se comprobó que cada nivel produjera un cambio observable en la velocidad.

## 5 Resultados

En esta práctica, se implementó un sistema de control de velocidad para un motor DC mediante PWM, utilizando un Arduino como generador de señales y la comunicación serial para ajustar los niveles de potencia. Los resultados obtenidos permitieron establecer una relación clara entre el ciclo de trabajo del PWM y el comportamiento del motor, lo que confirma la efectividad de esta técnica para el control de dispositivos electromecánicos.

Durante las pruebas, se observó que el motor comenzaba a girar de manera perceptible con un valor de PWM de 77 (donde 77 corresponde al valor mínimo encontrado experimentalmente). Este umbral representa el punto en el que el torque generado supera la fricción estática del motor. A medida que se incrementó el ciclo de trabajo, la velocidad del motor aumentó de forma aproximadamente lineal, como era de esperarse en un sistema PWM aplicado a un motor DC. Sin embargo, a partir de un cierto valor (255, en una escala de 0 a 255), la velocidad no se podía aumentar más.

Para una mejor visualización de los datos, se elaboró la siguiente tabla resumen:

Nivel	Valor PWM	Comportamiento Observado
<b>0</b>	0	Motor completamente detenido
<b>1</b>	77	Giro muy lento, apenas perceptible
<b>5</b>	166	Velocidad media, estable
<b>9</b>	255	Velocidad máxima

Además, se registró el siguiente comportamiento:

- **Respuesta del motor a cambios bruscos:** Al alternar rápidamente entre niveles bajos y altos, el motor mostró una respuesta inercial, con un ligero retraso en alcanzar la velocidad estable.

Estos resultados demuestran que, aunque el PWM es una técnica eficaz para el control de velocidad, su implementación práctica requiere considerar factores como la inercia mecánica, el ruido y los límites operativos del motor. La comunicación serial resultó ser un método efectivo para ajustar la potencia en tiempo real, facilitando la interactividad del sistema.

- Nivel 0

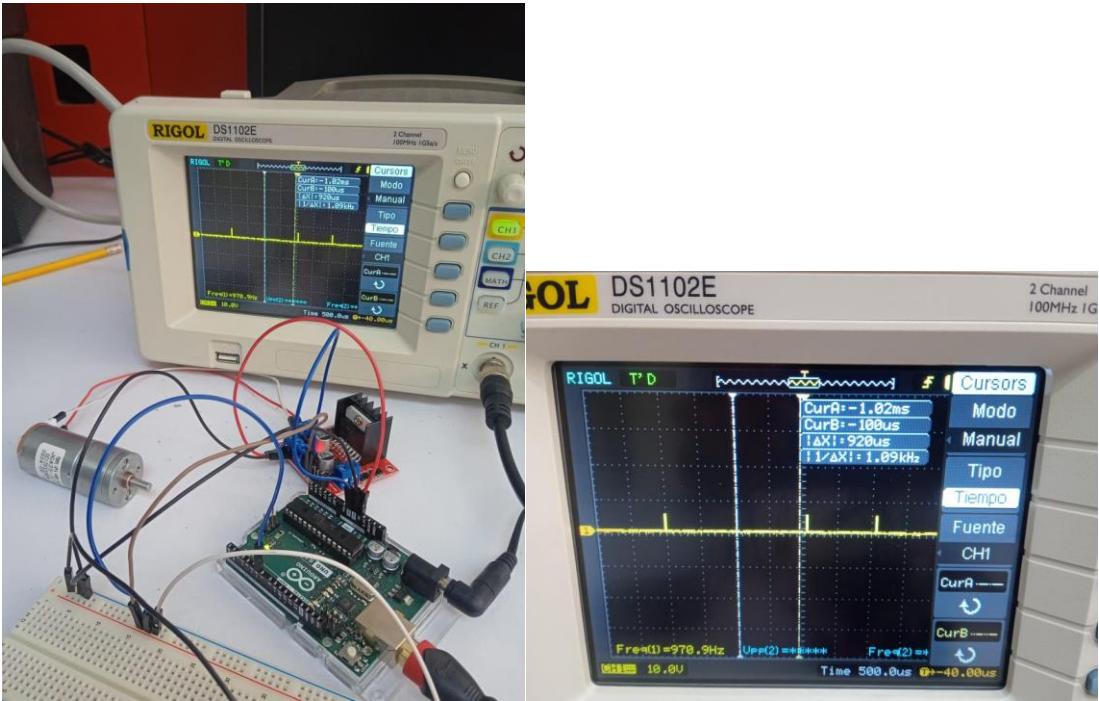


Figura 3 Nivel 0

- Nivel 1

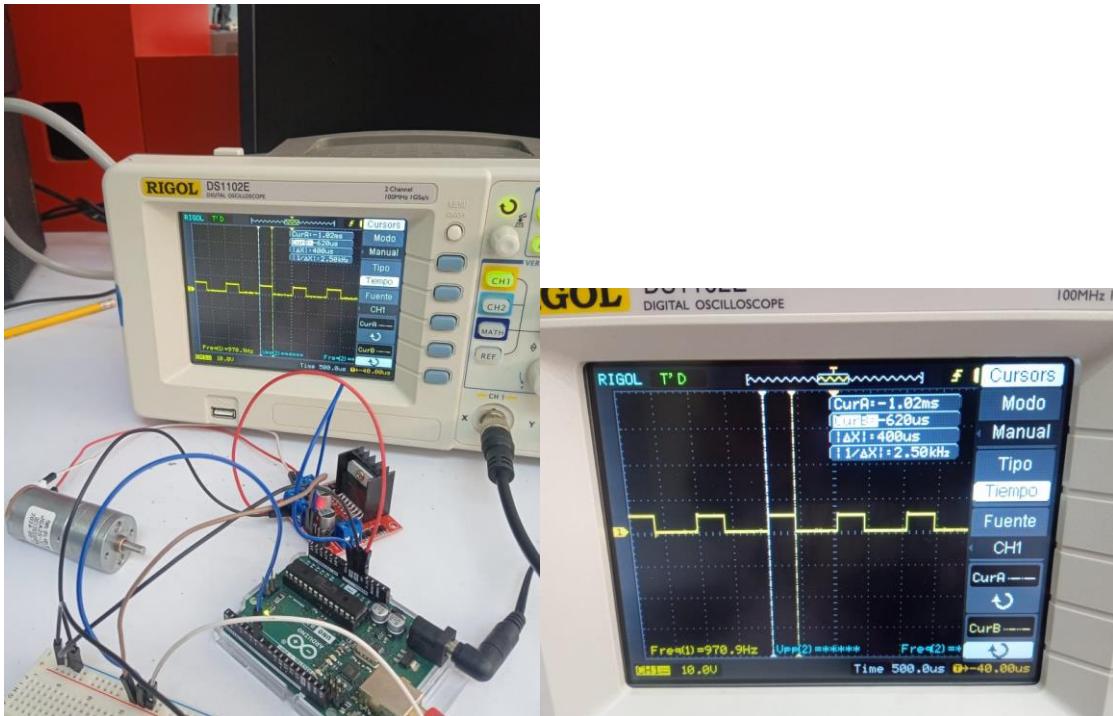


Figura 4 Nivel 1

- Nivel 2

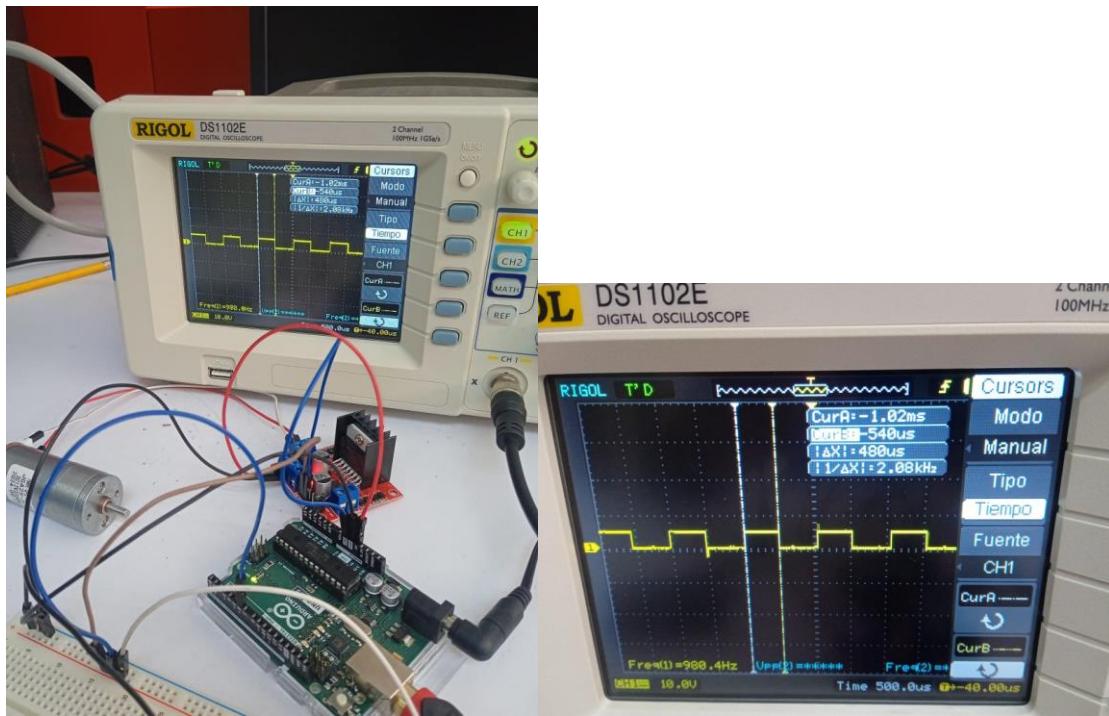


Figura 5 Nivel 2

- Nivel 3

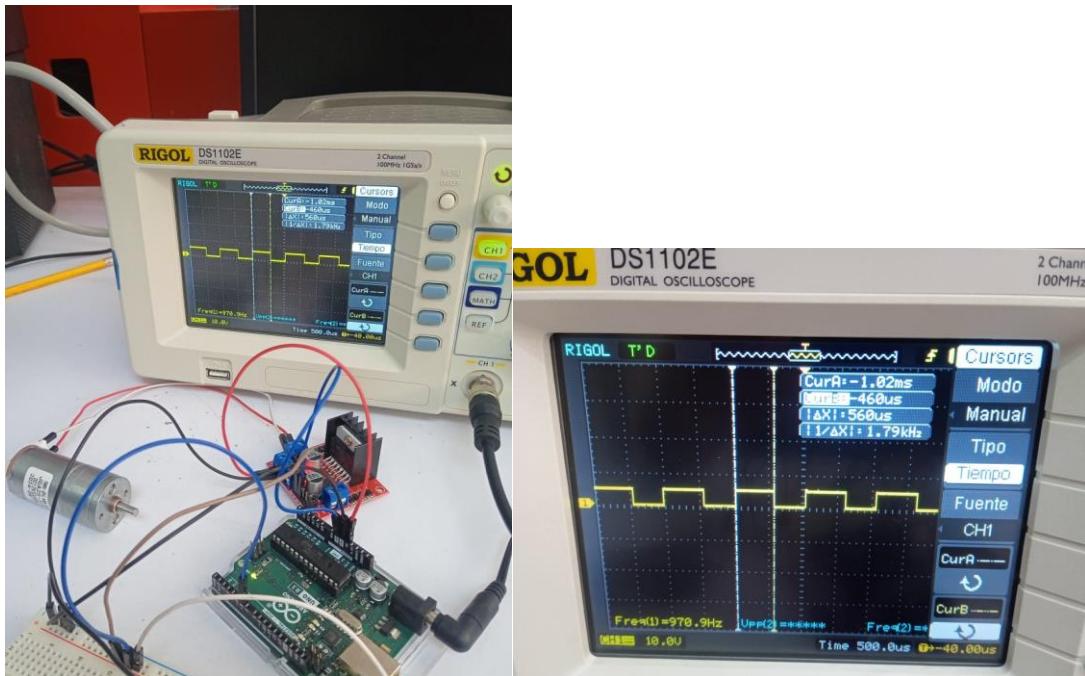
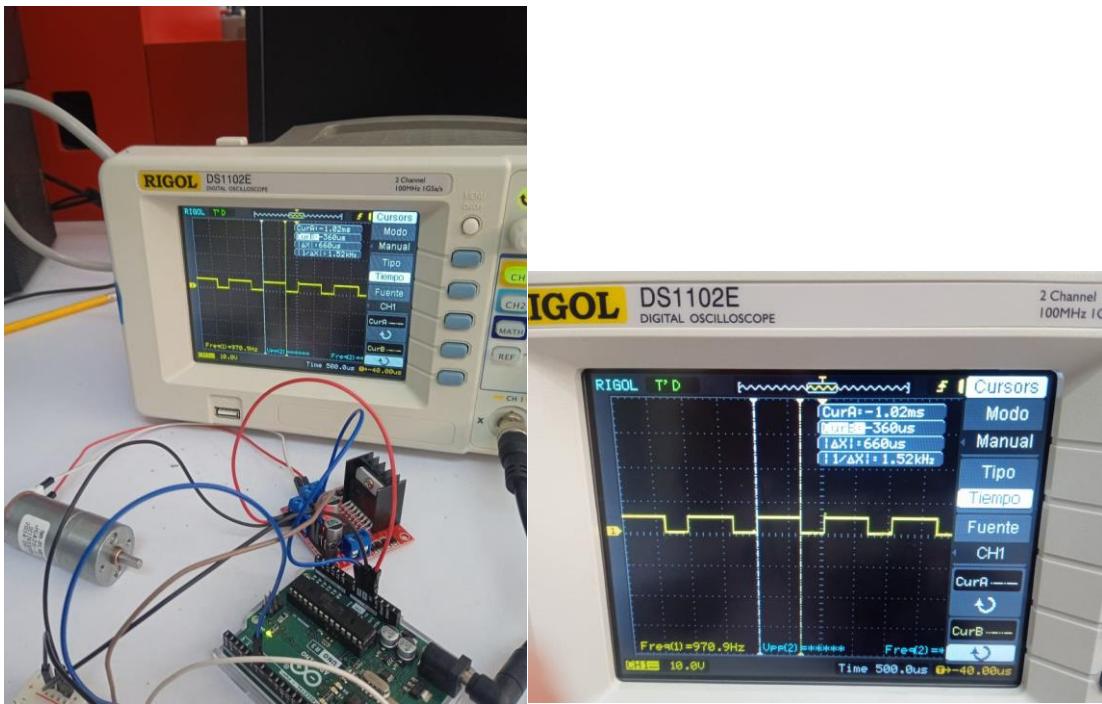


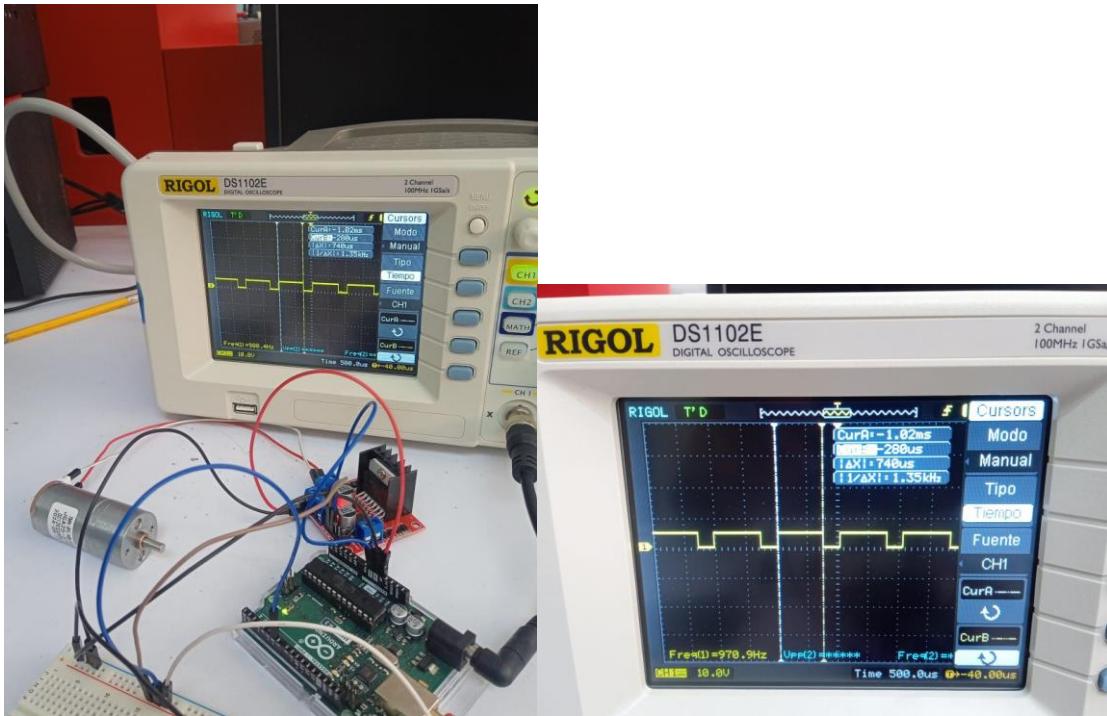
Figura 6 Nivel 3

- Nivel 4



*Figura 7 Nivel 4*

- Nivel 5



*Figura 8 Nivel 5*

- Nivel 6

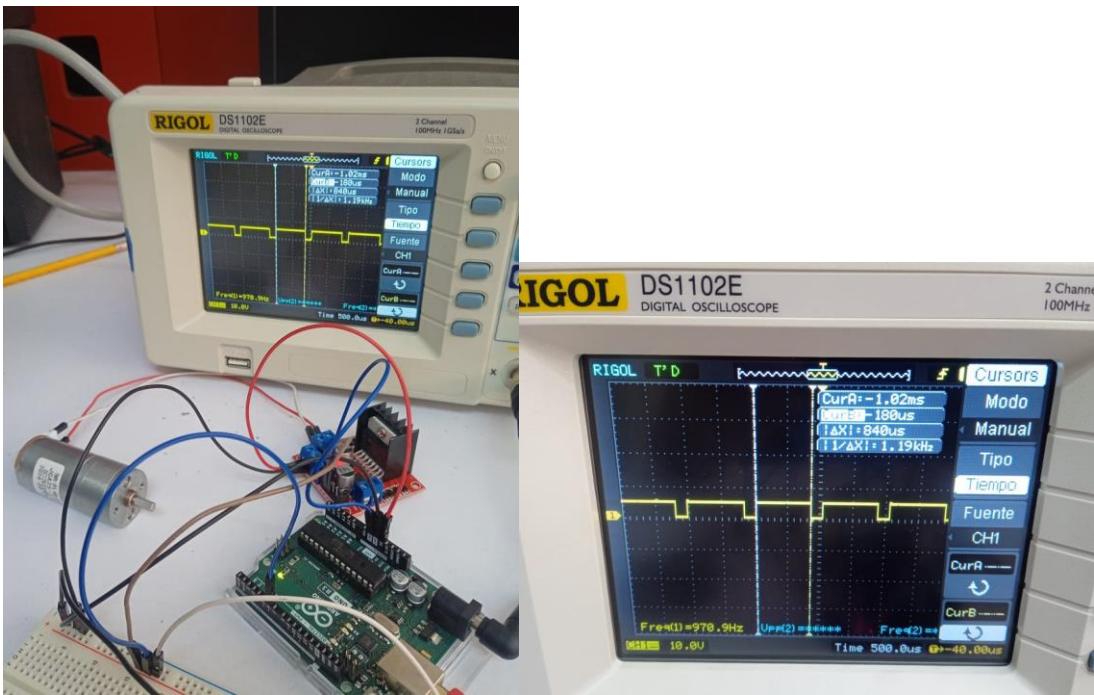


Figura 9 Nivel 6

- Nivel 7

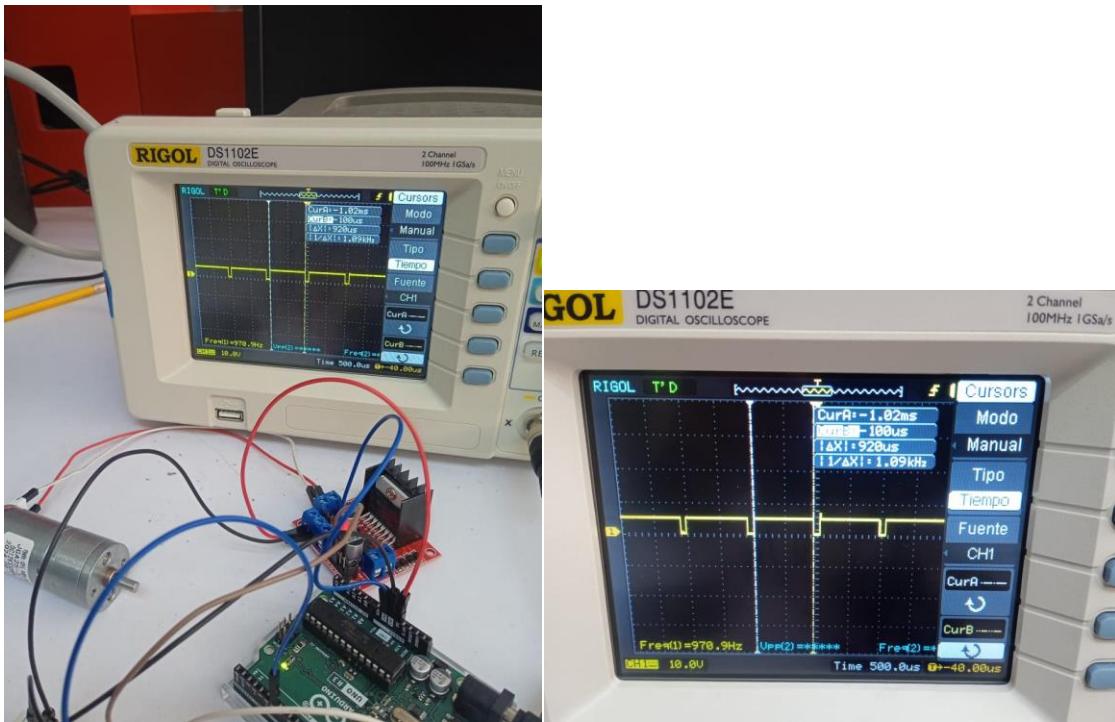


Figura 10 Nivel 7

- Nivel 8

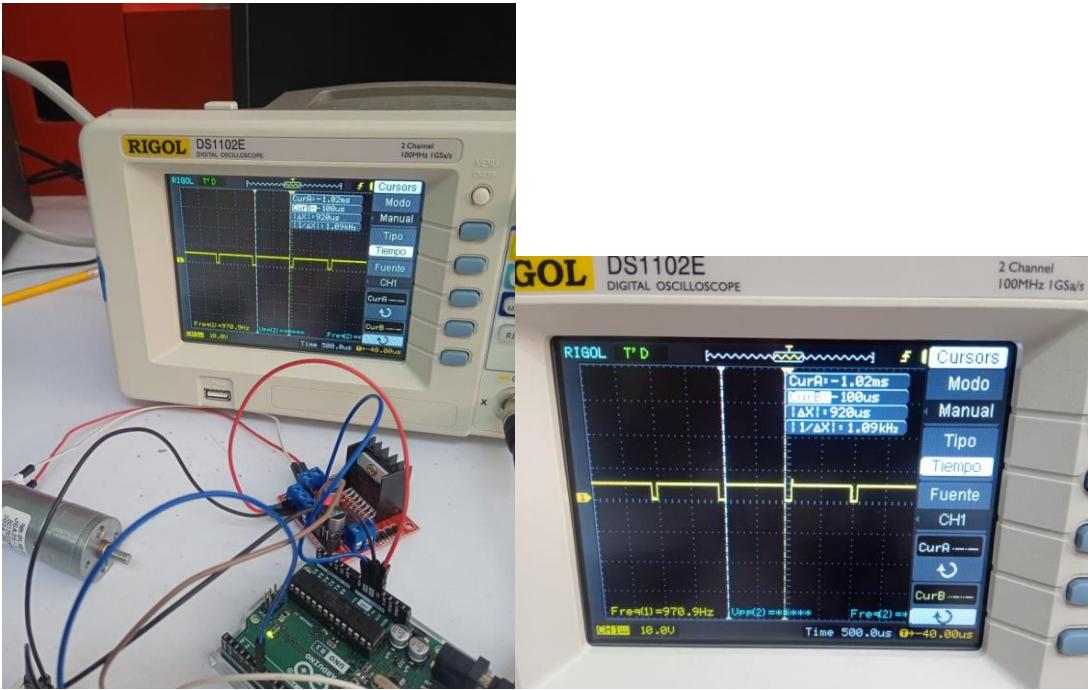


Figura 11 Nivel 8

- Nivel 9

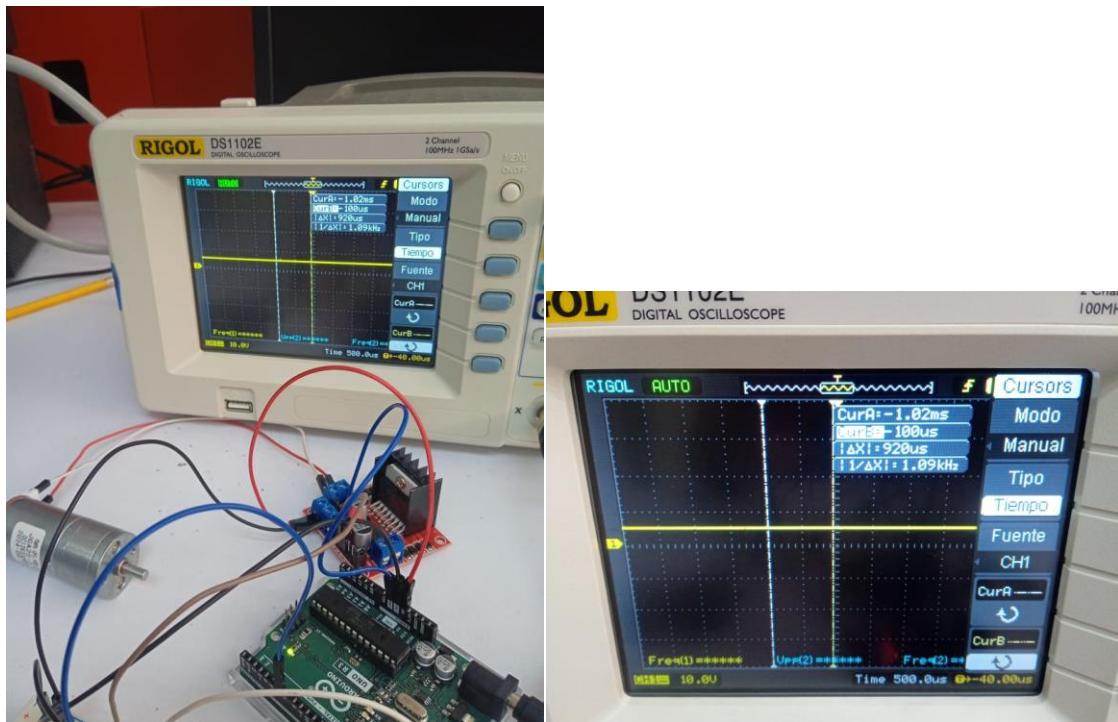


Figura 12 Nivel 9

## 5.1 Observaciones Clave

- El motor no respondía a valores de PWM inferiores al 30% ( $\approx 77$ ) debido a la fricción interna y la inercia.
- Se verificó que el driver L293D no introdujera distorsión en la señal PWM.

Este proceso permitió establecer una curva de control reproducible, esencial para aplicaciones como sistemas de ventilación o robótica.

## 6 Conclusiones

- Bejarano García Owen Uriel

El desarrollo de esta práctica permitió comprender de manera práctica el funcionamiento del PWM (Modulación por Ancho de Pulso) como técnica para controlar la velocidad de un motor DC. A través de la comunicación serial, se logró implementar un sistema de dosificación de potencia con 10 niveles predefinidos (del 0 al 9), donde cada nivel corresponde a un valor específico de PWM. Esto demostró cómo pequeños cambios en el ciclo de trabajo pueden alterar significativamente el comportamiento del motor, desde su encendido inicial hasta su velocidad máxima.

Uno de los hallazgos clave fue la identificación del umbral mínimo de PWM necesario para que el motor comience a girar, lo cual está relacionado con la fricción interna y la inercia del rotor. Asimismo, se observó que, al alcanzar ciertos valores altos de PWM, el motor ya no presentaba cambios perceptibles en su velocidad, lo que indica un límite en su respuesta a incrementos de potencia. Este comportamiento resalta la importancia de calibrar los rangos de trabajo según las características específicas del motor utilizado.

- García Quiroz Gustavo Iván

Entre las dificultades enfrentadas, destacó la necesidad de asegurar una alimentación estable para el motor, ya que fluctuaciones en la tensión afectaban directamente su rendimiento. Además, se verificó que el uso de un *driver* (como el L293D) fue esencial para proteger el Arduino de corrientes elevadas. Estas consideraciones prácticas refuerzan conceptos teóricos sobre el manejo de cargas inductivas y la selección adecuada de componentes en sistemas embebidos.

En aplicaciones reales, este principio puede extenderse a sistemas como:

- Control de velocidad en ventiladores o bombas.
- Robótica (regulación de movimiento en ruedas o brazos mecánicos).
- Sistemas de automatización industrial con ajuste fino de actuadores.



## 7 Referencias

- D. Features, “8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash,” Microchip.com. [Online]. Available: [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf). [Accessed: 25-Mar-2025].
- Atmel Corporation (2015). *ATmega328P Datasheet*. Secciones 14.9 (Registros de Timer/PWM) y 14.7 (Modos de Operación). Este documento proporcionó la base para configurar los registros TCCR1A/B y comprender los modos PWM del microcontrolador.
- Arduino LLC (2023). *Language Reference: analogWrite() y Serial Communication*. La documentación oficial de Arduino fue fundamental para implementar las funciones de comunicación serial y generación de señales PWM.